# CS4286: Assignment3

- Name: Zhang Deheng
- SID: 55199998

## 1. Security Services

- Zoom is the video-conferencing software widely used recently due to the outbreak of COVID-19[1]. “Zoombombing” is an unexpected issue during the conference by using Zoom. In this case, uninvited attendees break into and disrupt the meeting [2]. The attacker may join a session by using the shared password on the internet or simply using the link which allows legitimate users to join[1]. **The main failing services are access control and entity authentication.**
- Access control is compromised since the communication resource for a limited number of users is used by the attacker if they now the password, which should be protected by the access control mechanism. The technical vulnerability is that **there is no specific access control mechanism** for a session, anyone with a correct session password is allowed to access the conference.
- Entity authentication is compromised since there might be a masquerade situation. The zoom link for a legitimate user can be directly shared with the attackers with the password appended to the URL. In this case, if the attacker attends the conference through the shared link, the username presented in the meeting is the same as the legitimate user. The technical vulnerability is that **there is no freshness and data origin authentication provided**. The authentication is only based on the fixed password for each session by using challenge-and-response.
- Besides, **confidentiality and availability** might also be compromised since the conference content might be exposed and if the attackers cannot be removed, it will be a deny-of-conference-service.
- For access control, we can use the **access control model mechanism**. A role-based access control list can be constructed for each meeting. In this way, the host of the meeting can choose a white list of roles who can attend the meeting. For each role, user IDs are stored in a well-protected file. Therefore, attackers cannot attend the conference since they are not in the whitelist. Host can also store the users usually permitted to join the conference in a specific role list. For entity authentication, the mechanism **authentication protocol** should

be used. A nonce should be generated by the host and sent to the user who wants to join a session, and a response produced by MAC or encryption by a token stored locally. The token is generated and stored once the user is login into his/her zoom account. In this way, each time a user wants to join a session, entity authentication is provided.

## 2. Password File

- $y = H(s; password) \oplus password$: It is **secure** against precomputed dictionary attack since we cannot get the $password$ or $h(password)$ with a known $s$. We need to pre-compute all the possible $H(s; password) \oplus password$ to launch dictionary attack. Besides, hash and XOR are fast operations,m which is efficient for verification
- $y = H(s) \oplus H(password)$ : It is **insecure** against precomputed dictionary attack since if we know $s$, we can compute $H(password) = y \oplus H(s)$. Therefore, we can use the dictionary with pre-computed $H(password)$.
- $y = E_s(H(password))$ : It is **insecure** against precomputed dictionary attack since if we know $s$, we can compute $H(password) = D_s(y)$ where $D$ is the decryption of AES algorithm. Therefore, we can use the dictionary with pre-computed $H(password)$.
- $y = MAC_{password}(s)$ : It is **secure** against precomputed dictionary attack since we cannot get the $password$ or $h(password)$ with a known $s$. We need to pre-compute all the possible $MAC_{password}(s)$ to launch a dictionary attack.

## 3. TLS

- (a) There are 5 ciphersuites in the latest specification TLS 1.3 according to openssl_wiki.
    - TLS_AES_256_GCM_SHA384
    - TLS_CHACHA20_POLY1305_SHA256
    - TLS_AES_128_GCM_SHA256
    - TLS_AES_128_CCM_8_SHA256
    - TLS_AES_128_CCM_SHA256
- (b) Assume Alice ($A$) and Bob ($B$) are sharing a session key
    - TLS: the transport layer security protocol
    - AES_128_CCM: Authentication and encryption algorithm is 128-bit AES operating in Counter_CBC_MAC (CCM) mode.
    - SHA_256: The hash function is 256-bit SHA
    - The procedure of generating session keys and encrypting the data messages
        1. $A \rightarrow B : 5\ Cipher\ suites, TLS\ 1.3, R_A$
        2. $A \leftarrow B : TLS\_AES\_128\_CCM\_SHA256, Cert_B, R_B$
        3. $A \rightarrow B : \{S\}_B, SHA_{256}(SHA_{256}(msgs))$
        4. $A \leftarrow B : SHA_{256}(SHA_{256}(msgs))$
        5. $A \leftrightarrow B : AES\_128\_CCM_K(data)$
        - Where
            - $R_A$ and $R_B$ are nonces generated by $A$ and $B$ respectively
            - $Cert_B$ is the certificate of the public key of $B$
            - $\{\}_B$ represents the ciphertext encrypted by $B$'s public key

        - $S$ is randomly chosen by $A$
        - $K = SHA_{256}(S, R_A, R_B)$

## 4. Key agreement/IKE

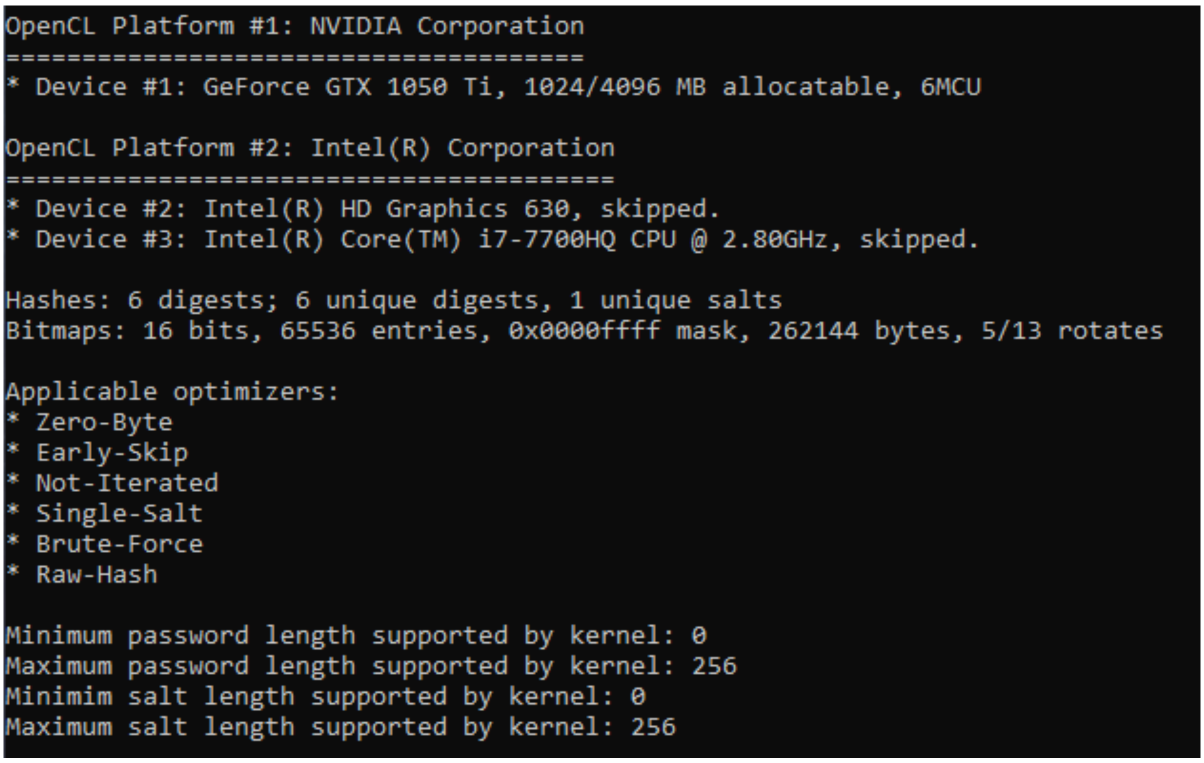
- (a) i. Authenticity and Key agreement
    - Authenticity: $A$ and $B$ are not authenticated to each other since there is no freshness included in the $proof_A$ and $proof_B$ and the nonce encrypted by the public key is not requred to be decrypted. **Once trudy knows Alice's and Bob's IP address**, he can intercept the message and lauch man-in-the-middle attack.
    - Key agreement: Key agreement is achieved since no one has the key control.
        - Key control: Neither entity of $A$ and $B$ has key control.
        - Key authentication: $g^{ab}\ mod\ p$ is only possible to be known by only one entity for both $A$ and $B$ despite themselves (which is an implicit key authentication).
            - However, according to the definition of key authentication, the entity that shares a key with you should be identified. If we define "identified" as "authenticated", it is neither an implicit nor an explicit key authentication.
- (a) ii. Modified protocol
    1. $A \rightarrow B : \{g^a\ mod\ p\}_{Bob}, \{\text{“}Alice\text{”}\}_{Bob}$
    2. $A \leftarrow B : \{g^b\ mod\ p\}_{Alice}, \{\text{“}Bob\text{”}\}_{Alice}, proof_B$
    3. $A \rightarrow B : proof_A$
    - Where
        - $proof_A = h(g^{ab}\ mod\ p, g^a\ mod\ p, g^b\ mod\ p, \text{“}Alice\text{”}, T_A)$
        - $proof_B = h(g^{ab}\ mod\ p, g^a\ mod\ p, g^b\ mod\ p, \text{“}Bob\text{”}, T_B)$
        - $K = h(g^{ab}\ mod\ p)$
        - $T_A$ and $T_B$ are timestamps generated by Alice and Bob respectively. (We assume they have synchronized clocks)
    - In this way authenticity between $A$ and $B$ is provided, where data origin authentication is provided by public key encryption, and freshness is provided by the timestamp. Key agreement is also provided.
- (b) This version is insecure because once the attacker Trudy ($T$) can intercept the first message, he can lauch a man-in-the-middle attack to establish a key with one of the participant $A$ while pretending to be $B$ as shown below.
    1. $A \rightarrow T : \text{“}Alice\text{”}, \text{“}Bob\text{”}, g^a\ mod\ p$
    2. $T \rightarrow B : \text{“}Trudy\text{”}, \text{“}Bob\text{”}, g^a\ mod\ p$
    3. $T \leftarrow B : \text{“}Bob\text{”}, \text{“}Trudy\text{”}, g^b\ mod\ p, [g^a\ mod\ p]_B$
    4. $A \leftarrow T : \text{“}Bob\text{”}, \text{“}Alice\text{”}, g^b\ mod\ p, [g^a\ mod\ p]_B$
    5. $A \rightarrow T : \text{“}Alice\text{”}, \text{“}Bob\text{”}, [g^b\ mod\ p, g^a\ mod\ p]_A$

## 5. IPSec

- (a) The tunnel mode should be used. Because even in ESP mode the original IP header is not encrypted and we can see who is the actual sender receiver is. Transport mode is designed for host-to-host instead of gateway-to-gateway. For tunnel mode, the header inside the intranet (e.g. 10.1.1.1) will be changed to a new header outside the intranet (e.g. 4.3.2.1). And by using ESP, confidentiality of the original header will be provided.
- (b) We name the data in the transport layer as packet, in the network layer as frame. The frame of each step is shown.
    1. Node 10.1.1.5 generates a frame with IP header (with source IP address 10.1.1.5 and target IP address 10.2.1.6), TCP header, and application layer data. The frame is sent to the intranet of 10.1.1.1.
    2. The router (10.1.1.1 - 4.3.2.1) GW encapsulate the frame adding a new IP header (with source IP address 4.3.2.1 and target IP address 8.7.6.5). Then,
        - if AH is used: authenticate all the immutable fileds in the new IP header as well as the original frame.
        - if ESP is used: authenticate and encrypt the original frame, append the ESP auth field.
    3. The router (10.2.1.1 - 8.7.6.5) GW de-capsulate the frame (strip extra headers) and inject the frame to the intra net of 10.2.1.1. The detail process can be divided into two cases:
        - if AH is used: verify all the immutable fileds in the new IP header as well as the original frame. Remove the new IP header and the AH header.
        - if ESP is used: verify and decrypt the original frame. Remove the ESP auth field, new IP header, and the ESP header.
    4. Node 10.2.1.6 receives the frame.
    - Note that the routing tables for the two routers needs to direct the frames to the GWs so these could be directed.

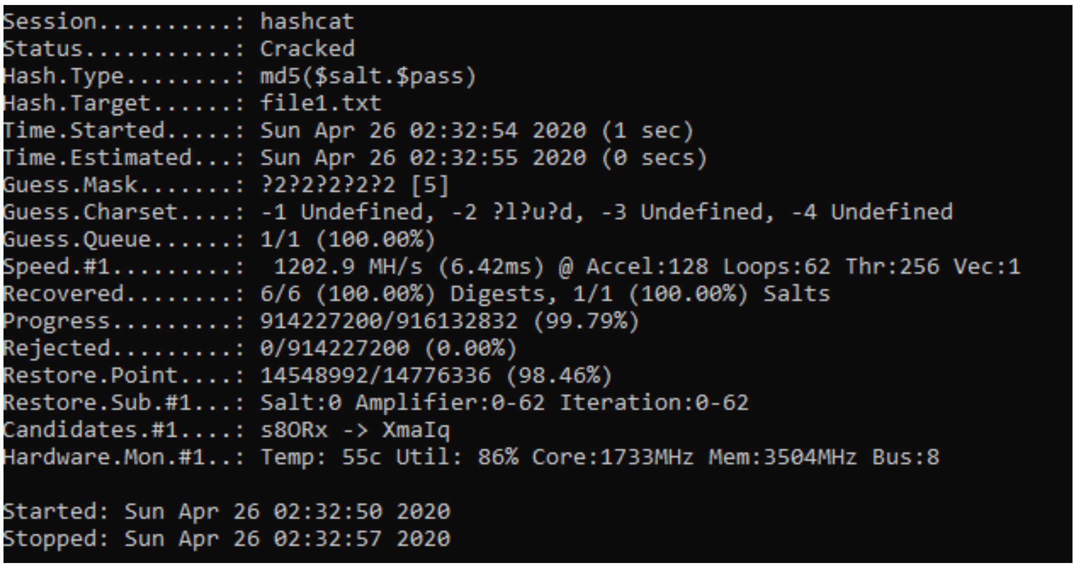
## 6. Password File

- Following snapshot shows the environment of this experiment



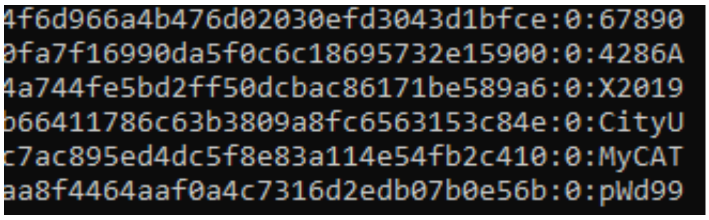
```
OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1050 Ti, 1024/4096 MB allocatable, 6MCU

OpenCL Platform #2: Intel(R) Corporation
========================================
* Device #2: Intel(R) HD Graphics 630, skipped.
* Device #3: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, skipped.

Hashes: 6 digests; 6 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Applicable optimizers:
* Zero-Byte
* Early-Skip
* Not-Iterated
* Single-Salt
* Brute-Force
* Raw-Hash

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimim salt length supported by kernel: 0
Maximum salt length supported by kernel: 256
```

- (a)
  - File 1
    - Command: `hashcat64.exe -m 20 -a 3 file1.txt -1 ?l?u?d ?1?1?1?1?1`
      - `-a 3`: brute force attack
      - `-m 20`: Use the mode `md5($salt.$pass)`
      - `?1?1?1?1?1`: Five characters from charset one includes characters `?l?u?d`
    - Time:



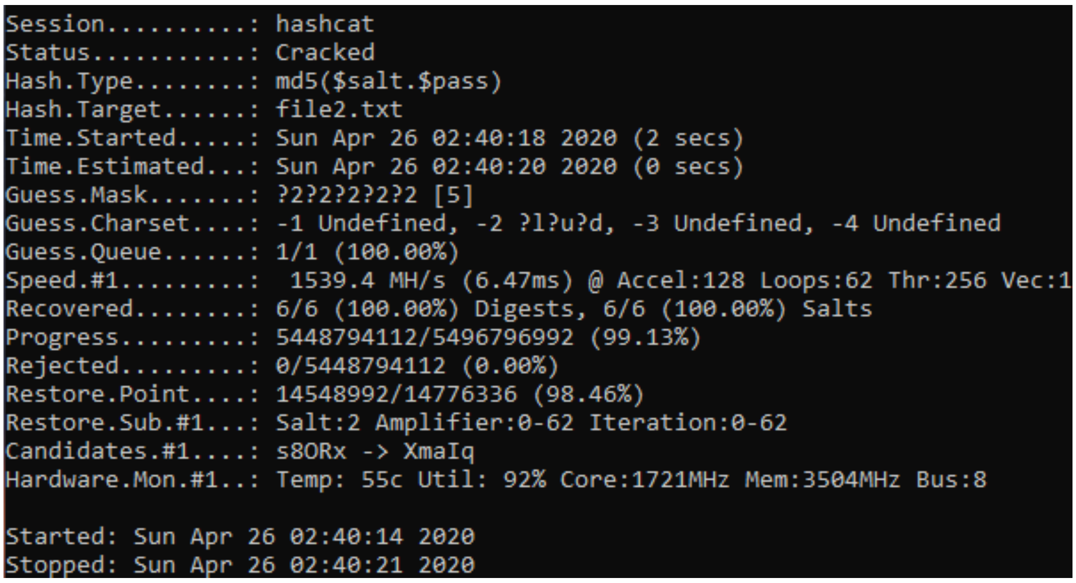
```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: md5($salt.$pass)
Hash.Target......: file1.txt
Time.Started.....: Sun Apr 26 02:32:54 2020 (1 sec)
Time.Estimated...: Sun Apr 26 02:32:55 2020 (0 secs)
Guess.Mask.......: ?2?2?2?2?2 [5]
Guess.Charset....: -1 Undefined, -2 ?l?u?d, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1202.9 MH/s (6.42ms) @ Accel:128 Loops:62 Thr:256 Vec:1
Recovered........: 6/6 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 914227200/916132832 (99.79%)
Rejected.........: 0/914227200 (0.00%)
Restore.Point....: 14548992/14776336 (98.46%)
Restore.Sub.#1...: Salt:0 Amplifier:0-62 Iteration:0-62
Candidates.#1....: s8ORx -> XmaIq
Hardware.Mon.#1..: Temp: 55c Util: 86% Core:1733MHz Mem:3504MHz Bus:8

Started: Sun Apr 26 02:32:50 2020
Stopped: Sun Apr 26 02:32:57 2020
```

    - Result:



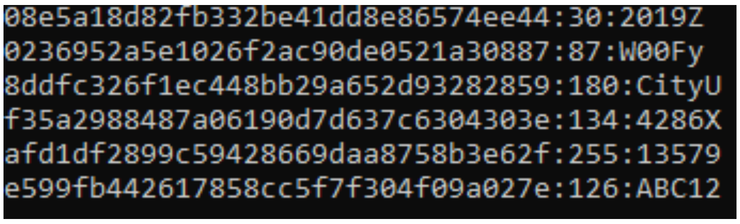
```
4f6d966a4b476d02030efd3043d1bfce:0:67890
0fa7f16990da5f0c6c18695732e15900:0:4286A
4a744fe5bd2ff50dcbac86171be589a6:0:X2019
b66411786c63b3809a8fc6563153c84e:0:CityU
c7ac895ed4dc5f8e83a114e54fb2c410:0:MyCAT
aa8f4464aaf0a4c7316d2edb07b0e56b:0:pWd99
```

- File 2
  - Command: `hashcat64.exe -m 20 -a 3 file2.txt -1 ?l?u?d ?1?1?1?1?1`
    - `-a 3`: brute force attack
    - `-m 20`: Use the mode `md5($salt.$pass)`
    - `?1?1?1?1?1`: Five characters from charset one includes characters `?l?u?d`
  - Time:



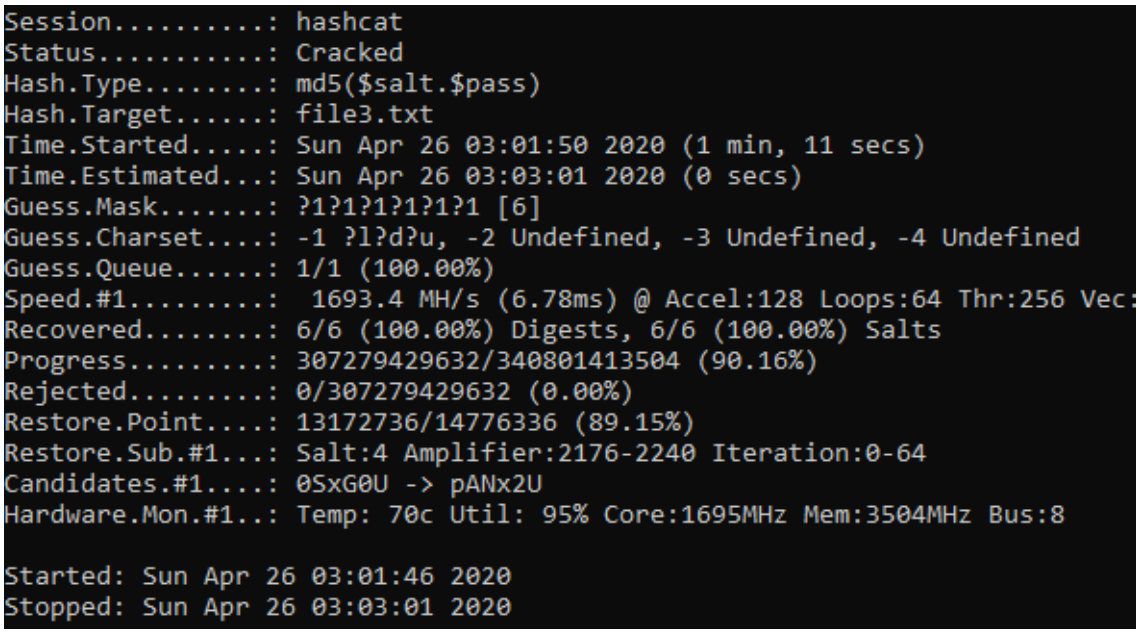
```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: md5($salt.$pass)
Hash.Target......: file2.txt
Time.Started.....: Sun Apr 26 02:40:18 2020 (2 secs)
Time.Estimated...: Sun Apr 26 02:40:20 2020 (0 secs)
Guess.Mask.......: ?2?2?2?2?2 [5]
Guess.Charset....: -1 Undefined, -2 ?l?u?d, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1539.4 MH/s (6.47ms) @ Accel:128 Loops:62 Thr:256 Vec:1
Recovered........: 6/6 (100.00%) Digests, 6/6 (100.00%) Salts
Progress.........: 5448794112/5496796992 (99.13%)
Rejected.........: 0/5448794112 (0.00%)
Restore.Point....: 14548992/14776336 (98.46%)
Restore.Sub.#1...: Salt:2 Amplifier:0-62 Iteration:0-62
Candidates.#1....: s8ORx -> XmaIq
Hardware.Mon.#1..: Temp: 55c Util: 92% Core:1721MHz Mem:3504MHz Bus:8

Started: Sun Apr 26 02:40:14 2020
Stopped: Sun Apr 26 02:40:21 2020
```

  - Result:



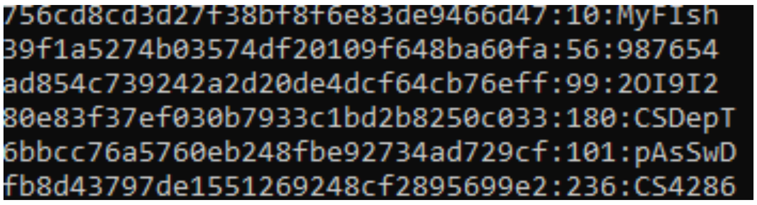
```
08e5a18d82fb332be41dd8e86574ee44:30:2019Z
0236952a5e1026f2ac90de0521a30887:87:W00Fy
8ddfc326f1ec448bb29a652d93282859:180:CityU
f35a2988487a06190d7d637c6304303e:134:4286X
afd1df2899c59428669daa8758b3e62f:255:13579
e599fb442617858cc5f7f304f09a027e:126:ABC12
```

- (b)
  - command: `hashcat64.exe -m 20 -a 3 file3.txt -1 ?l?d?u ?1?1?1?1?1?1`
    - `-a 3`: brute force attack
    - `-m 20`: Use the mode `md5($salt.$pass)`
    - `?1?1?1?1?1?1`: Six characters from charset one includes characters `?l?u?d`
  - Time:



```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: md5($salt.$pass)
Hash.Target......: file3.txt
Time.Started.....: Sun Apr 26 03:01:50 2020 (1 min, 11 secs)
Time.Estimated...: Sun Apr 26 03:03:01 2020 (0 secs)
Guess.Mask.......: ?1?1?1?1?1?1 [6]
Guess.Charset....: -1 ?l?d?u, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1693.4 MH/s (6.78ms) @ Accel:128 Loops:64 Thr:256 Vec:
Recovered........: 6/6 (100.00%) Digests, 6/6 (100.00%) Salts
Progress.........: 307279429632/340801413504 (90.16%)
Rejected.........: 0/307279429632 (0.00%)
Restore.Point....: 13172736/14776336 (89.15%)
Restore.Sub.#1...: Salt:4 Amplifier:2176-2240 Iteration:0-64
Candidates.#1....: 0SxG0U -> pANx2U
Hardware.Mon.#1..: Temp: 70c Util: 95% Core:1695MHz Mem:3504MHz Bus:8

Started: Sun Apr 26 03:01:46 2020
Stopped: Sun Apr 26 03:03:01 2020
```

- Result:



```
756cd8cd3d27f38bf8f6e83de9466d47:10:MyFIsh
39f1a5274b03574df20109f648ba60fa:56:987654
ad854c739242a2d20de4dcf64cb76eff:99:2OI9I2
80e83f37ef030b7933c1bd2b8250c033:180:CSDepT
6bbcc76a5760eb248fbe92734ad729cf:101:pAsSwD
fb8d43797de1551269248cf2895699e2:236:CS4286
```

- (c) The time for recovering file1 and file2 are 1 seconds and 2 seconds respectively. However, if we **use CPU**, file1 takes around 70 seconds while file2 takes around 230 seconds. Although the ratio time taken by file2 over time taken by file1 is 3.28 instead of 5, it supports the theory since file2 takes longer time than file1 considering the error during the experiment.
- (d) To recover file3, the time needed is significantly increased to 71 seconds by using GPU, which is around 35 times longer than file2, 70 times longer than file1.